VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Rozšíření programu UniSave o novou funkcionalitu

Implementation of New Functionality to Program UniSave

2013 **Michal Vláčil**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Vláčil**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Rozšíření programu UniSave o novou funkcionalitu
Implementation of New Functionality to Program UniSave

Zásady pro vypracování:

Cílem práce je rozšířit stávající program UniSave. Program UniSave je implementován v jazyce Java a slouží pro načítání hodnot z různých měřidel. Tyto naměřené hodnoty se dále zpracovávají a vytváří se z nich protokol o měření.

Program bude rozšířen o:
1. Kombinovanou podporu ukládání do XML souborů a do databáze.
2. Úprava editoru pro šablony měření.

Práce bude obsahovat:
1. Implementaci výše uvedených rozšíření.
2. Popis rozšíření v jazyce UML.
3. Dokumentaci.

Seznam doporučené odborné literatury:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Hranicích dne 21.7.2013 .....

Poděkování

Chtěl bych poděkovat svému vedoucímu práce Ing. Davidu Ježkovi,Ph.D., za jeho čas při konzultacích a posktynutí důležitých rad pro úspěšné zvládnutí práce. Dále děkuji své rodině za vytvoření dobrých studijních podmínek nejen při psaní bakalářské práce, ale také po celou dobu studia na vysoké škole.

Abstrakt

Cílem této bakalářské práce je upravení stávajícího programu Unisave, aby měl možnost ukládat data do XML souborů a zároveň do databáze přes aplikační a perzistentní vrstvu. Dalším bodem je upravení editoru šablon, kvůli možnosti použít novější knihovnu pro zpracování protokolů. Program UniSave je napsán v jazyce Java a slouží k zaznamenání naměřených hodnot z různých měřících zařízení. Tyto hodnoty a i nastavení programu by měly být úkládány po volbě uživatele buď do XML nebo databáze .

Klíčová slova

Java, Hibernate, objektově-relační mapování, perzistentní vrstva,

UniSave,XML, HTML

Abstract

The goal of this Bachelor thesis is to customize current program Unisave, to have an opportunity to save data to XML files and at the same time to database through application persistent layer. Next point is to customize the template editor, to allow it using newer library for processing the protocols. Program Unisave is written in Java language and is used to record measured values from various measuring devices. These values and even program options should be saved after the user picks a choice to save to |XML or database.

Key words

Java, Hibernate, object-relational mapping, persistent layer,

UniSave, XML, HTML

Seznam použitých symbolů a zkratek

DOM – Document Object Model

SAX – Simple API for XML

JAXB – Java Architecture for XML Binding

HTML – HyperTExt Markup Language

GUI – Graphical User Interface

XML – eXtensible Markup Language

Seznam obrázků

Seznam zdrojových kódů

Obsah

1 Úvod

Program UNISAVE je implementován pomocí jazyka Java. Zpracovává naměřená data z různých druhů měřidel. Tato měřidla se dají připojit k počítači pomocí sériového či infračerveného portu nebo je také možno vkládat data ručně. Prozatím program UNISAVE uměl pracovat jen s XY měřením, ukládat data do databáze a do XML pomocí metody SAX.

Cílem této práce je rozšířit možnosti programu o nový druh měření, přesněji Statistické měření, doplnit oběma těmto měřením podporu ukládání do XML a databáze. Dát uživateli možnost volby jestli se data budou načítat a ukládat do databáze nebo XML. Dalším bodem je upravení Report Designeru, který vytváři šablony pro protokoly měření, aby vytvářel správný druh XML protokolů pro použití s novou knihovnou na zpracování protokolů.

Následující kapitoly popisují postup při tvoření práce. Problematiku při změnách ukládání a načítání dat, problémy, které vznikly přidáním nových tříd pro měření, úpravy programu, aby dokázal pracovat s přidaným kódem a funkcemi. Jde o úpravy GUI a objektů Action, práce s databází, zpracování XML a HTML, úpravu nástrojových lišt a menu. V první části budou zjednodušeně vysvětleny některé použité technologie, se kterými jsem se setkal při zpracovávání práce.

2 Přehled použitých technologií

2.1) JAVA

Už od roku 1991 firma Sun Microsystems vyvíjela programovací jazyk na principech C a C++ pro "vestavěné systémy", což je odborný termín pro běžná elektronická zařízení ovládaná zabudovaným mikroprocesorem. Původní název byl Oak, neboli dub, který stál před oknem vedoucího týmu, pana Goslinga.

Nastal ale problém, protože programovací jazyk s tímto jménem již existoval, takže vývojová skupina hledala nový název pro svůj jazyk. Název vznikl náhodou, když skupina navštívila firemní bufet kde padl nápad, dát jazyku jméno podle amerického slangového slova Java, které znamená "kafe". Projektu se zpočátku moc nedařilo, ale v roce 1993 si firma Sun včas uvědomila důležitost WWW (World Wide Web) na trhu a využití jejich jazyka pro programování webových aplikací. O dva roky pozdější byla Java oficiálně představena firmou Sun na konferenci. Od této konference se stala Java nedílnou součástí světa programovacích jazyků a měla v budoucnu hrát velkou roli na tomto trhu.

Java díky své přenositelnosti je používaná pro programy, které mají pracovat na různých systémech. Od možnosti použití práce s čipovými kartami (platforma JavaCard) nebo mobilními zařízeními (platforma Java ME), aplikacemi pro desktopové počítače (platforma Jaa SE) až po možnosti v rozsáhlých distribuovaných systémech na spolupracujících počítačích rozprostřených po celém světě (platforma Java EE). Tyto technologie se jako celek nazývají platforma Java.[1]

Java, kromě osmi primitivních datových typů, je plně objektově orientovaný jazyk. Java má také nevýhody. Spuštění programů v Javě je oproti jazyku C++, který provadí tzv. statickou kompilaci, o dost pomalejší, protože prostředí musí program napřed přeložit a až poté spustit. Další nevýhodou bývá větší paměťová náročnost při běhu programu.

8. května 2007 firma Sun uvolnila zdrojové kódy Javy, dále bude vyvíjena jako open source. V červenci roku 2011 vyšla nejnovější verze 7, kde produkční verze má označení 7.0 a vývojářská 1.7.0.

Řekněme si něco o klíčových vlastnostech Javy. První je již výše zmíněná podpora objektově orientovaného programování. Oproti jazykům, jako je třeba C++, již neumožňuje napsat program, který by nebyl objektově orientovaný. Je jistá možnost, jako v jiných jazycích, psát neobjektově, ale to je většinou na samotném programátorovi jak bude kód psát.[4]

Další vlastnost Javy je její jednoduchost. Základy pochopení její syntaxe pro člověka, co umí programovat, zabere jen pár hodin. Jednoduchost si ale nesmíme spojovat s tím, že by byl nějak omezený.

Java je multiplatformní. Řadí se mezi hybridní jazyky, což znamená, že je současně překládaná i intepretována. Program napsaný v jazyce Java se napřed přeloží do speciálního tvaru, kterému se říká bajtkód, který je následně zanalyzován a interpretován speciálním programem nazvaným virtuální stroj Javy. Tento virtuální stroj umožňuje, aby stejný program fungoval na různých počítačích s odlišnými operačními systémy. Každá konfigurace hardwaru a operačního systému si může definovat vlastní virtuální stroj. Ten se pak sám stará o hladký chod programu. Java funguje na všech běžných používaných operačních systémech.

2.2) Relační databáze

Relační databáze vychází z relačního modelu. Tímto pojmem se mnohdy označuje mimo databázi i samotné softwarové řešení. Relační databáze je založena na tabulkách, které mají řádky a sloupce, kde každý řádek je nový záznam dat pro jednotlivé atributy které jsou reprezentovány sloupci. Termín relační databáze definoval Edgar F. Codd už v roce 1970.

Základní konstruktor RDB je relace, neboli databázová tabulka, což je dvourozměrná struktura se záhlavím a tělem. Její sloupce jsou atributy a řádky záznamy. Atributy mají svůj datový typ a doménu, což je výčet přípustných hodnot pro daný datový typ. Každý záznam musí mít svůj primární klíč, který je jednoznačný identifikátor záznamu. Nesmí být nikde použit dvakrát stejný klíč. Klíč bývá vetšinou jeden atribut nebo více spojených pro přesné určení jedinečnosti. Tabulka obsahuje i cizí klíče pro použití asociací s jinými tabulkami. Pojem integrita databáze znamená, že uložená data jsou konzistentní vůči definovaným pravidlům. Tabulka splňuje požadavky datových typů, IO a domén.

2.3) Hibernate

Otcem Hibernate je Gavin King, který posléze přešel pod formu JBoss a ta byla v roce 2006 převzata firmou Red Hat, která i v současné době pokračuje na vývoji frameworku Hibernate.

Program UNISAVE teď používá pro ukládání ORM Hibernate, který umožňuje tvořit perzistentní třídy podle objektově orientovaného idiomu, který obsahuje asociace mezi třídami, polymorfismus, dědičnost, kolekce apod. Hibernate je možné využívat jak na architekturách Java i .NET, ale i v běžných aplikacích a webových aplikacích. Aby Hibernate věděl jak má třídy ukládat, používá tzv. mapovací soubory, ve kterých je popsáno jak budou data z programu uložena nebo načtena z databáze.

Mapovací soubor je ve formátu xml s příponou "hbm.xml". Vždy používá jeden mapovací soubor pro každou třídu, kterou chceme pomocí Hibernate ukládat. Většinou jsou tyto soubory ve stejném adresáři jako samotné třídy, ale není to povinné.

Ukázka kódu mapovacího souboru :

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="unisave2006">
<class name="PortName">
<id name="id" column="PORT_ID">
<generator class="native"></generator>
</id>
<property name="name"></property>
<property name="fileName"></property>
<property name="portIndex"></property>
</class>
</hibernate-mapping>
```

Kód 1 : Mapovací soubor Hibernate

Hibernate potřebuje pro svou funkci několik nastavení. Například údaje o JDBC driveru, typ databáze, ale i všechny cesty k mapovacím souborům.

2.4) XML

Původně měl program UNISAVE jen podporu ukládání a načítani přes XML. K této možnosti ukládání se v rámci bakálářské práci vratíme. Co ale samotný jazyk XML je a v jakém zpracování se pro náš program vyskytuje.

XML je rozšiřitelný značkovací jazyk, který byl vyvinut a zestandardizován World Wide Web Consortiem, zkráceně W3C. Jde o zjednodušený jazyk oproti jeho předchůdci SGML. Používá se hlavně pro uchovávání dat, jejich přenos nebo serializaci. XML má podporu ve většině programovacích jazycích. XML byl vytvořen jako univerzální možnost přenosu dat mezi programy, aby se nemuselo složitě zpracovávat nepřeberné množství formátů jako jsou DOC, XLS a jiné. XML specifikace od W3C je zdarma a každý si může do svých programů implementovat podporu XML.

XML má dobrou mezinárodní podporu. Už od jeho vytváření se myslelo i na jiné jazyky než byla angličtina. Používá se implicitně znaková sada ISO 10646, díky tomu můžeme v XML vytvářet dokumenty, které obsahují více jazyků. XML podporuje i jiné druhy kódování.

XML poskytuje vysoký informační obsah díky upuštění od značkování zaměřeného na prezentaci. XML neobsahuje předdefinované tagy, ale máme možnost si je definovat v souboru DTD (Document Type Definition). Potom si můžeme XML automaticky kontrolovat, jestli odpovída námi vytvořené definici. DTD není jediný definiční jazyk, ale W3C pracuje na jeho sjednocení.

2.4.1) Struktura XML

Samotná struktura XML je velmi jednoduchá. Skládá se ze dvou částí, hlavičky a těla dokumentu. V hlavičce jsou nezbytné údaje pro interpretaci obsahu těla dokumentu. Tělo dokumentu obsahuje vlastní data, rozdělené do elementů, které jsou definované stejným začátečním a koncových tagem (viz. kód 2).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <kniha>
        <název>Pán Prstenů</název>
        <cena>356</cena>
        <autor> Tolkien</autor>
        <zanr>Fantasy</zanr>
    </kniha>
```

Kód 2 : Ukázka XML

2.4.2) XML a JAVA

Java má několik možností, jak pracovat s XML. Jednou metodou je SAX, kterou používá program UNISAVE a kterou popíšu blíže níže. Další možností je DOM a JAXB.

2.4.3) SAX

SAX není standard W3C, ale vytvořila ho skupina lidí kolem konference XML-DEV. SAX nemá ambice nahradit DOM. SAX i DOM se dobře doplňují. DOM pro vytvoření svého stromu používá právě SAX. SAX parser při procházení dokumentu průběžně vyvolává eventy, které může programátor odchytávat pomocí SAX api. SAX je vhodnější pro velké dokumenty, protože je nemusí ukládat do paměti. Pomocí SAX se můžeme zaměřit jen na část dokumentu a z něj vytvořit strom. SAX je rychlejší než DOM. SAX má i pár nevýhod, nemůžeme s ním dokument upravovat, nemůžeme se vrátit zpět ke zpracovaným datům, proto je práce se stromem pohodlnější.

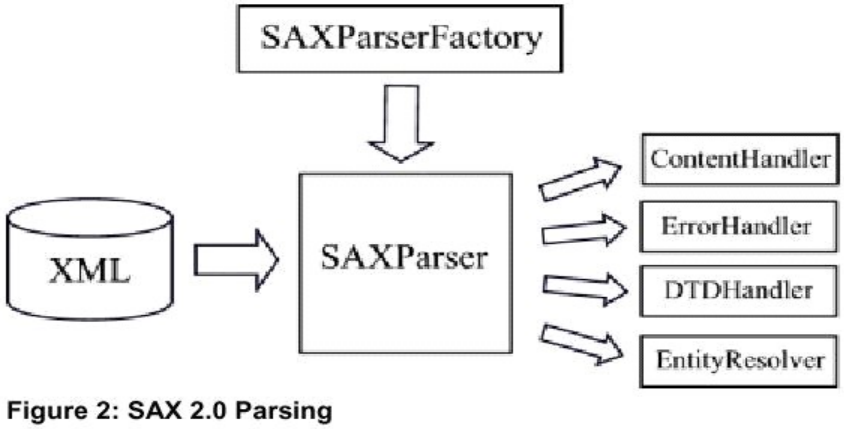


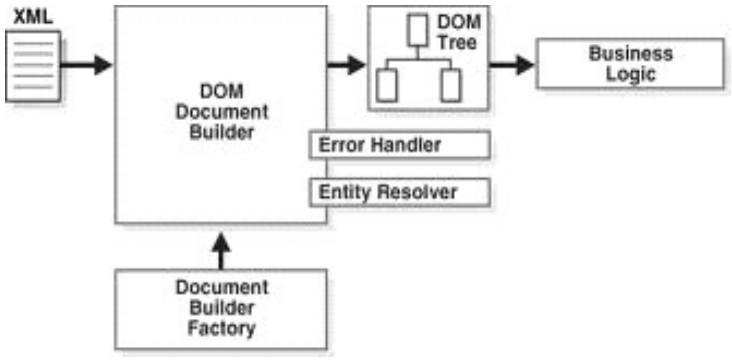
Figure 2: SAX 2.0 Parsing

Obrázek 1: Model SAX

2.4.4) DOM

DOM je postaveno na jiném principu než SAX. DOM je prezentovaný stromovou strukturou, kdy každý element odpovídá jednomu uzlu stromu. Odpovídající uzly mají i komentáře, instrukce pro zpracování atd. Rozhraní DOM obsahuje funkce pro procházení stromu, modifikaci, mazaní nebo přidávání uzlů. Narozdíl od SAXU se nemusí dokument procházet od začátku do konce, ale pohybovat se můžeme jak potřebujeme. DOM je využitelný u programů, které provádějí náročnější operace s

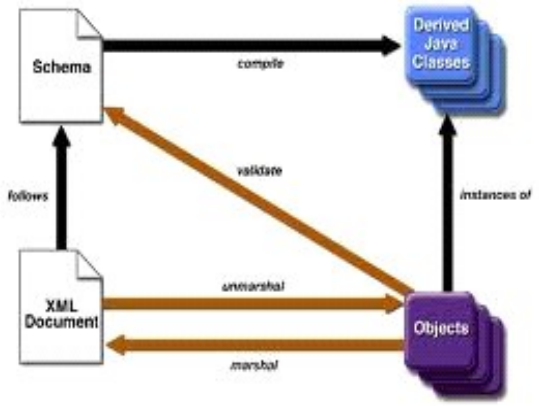
dokumentem. Rozhraní DOM je standardem od W3C. Původně bylo vytvořeno zejména proto, aby prohlížeče podporující XML používaly stejný objektový model pro přístup k dokumentu. Tento standard používají prohlížeče Internet Explorer a Mozilla Firefox.



Obrázek 2 : Model DOM

2.4.5) JAXB

JAXB nabízí možnost konvertovat java objekty na xml a naopak, umožňuje zápis a čtení XML z různých zdrojů, například ze souboru, streamu nebo URL. U JAXB oproti SAX není potřeba vytvářet parser nebo metody zpětného volání. Stromy vytvořené pomocí JAXB jsou efektivnější na paměť než je tomu u technologie DOM.



Obrázek 3 : Model JAXB

3 Vlastní práce na programu Unisave

3.1) Ukládání a načítání měření do XML

Ukládání měření je řešeno ve třídě *Measurement* v metodě *SaveToXML()*. Tato třída ukládala měření pomocí metody SAX. Podle požadavků bakalářské práce jsem změnil použití technologie na JAXB. Pro správnou funkci ukládání přes JAXB je potřeba do třídy napsat anotace pro třídu, její metody a atributy. Důležité je provést importy knihoven pro práci s JAXB. U třídy *Measurement* je problém že je abstraktní a tudíž nejde převést přímo pomocí JAXB. Musíme proto u jejích potomků vytvořit jejich vlastní anotace pro zpracování pomocí JAXB.

```
@XmlAccessorType(XmlAccessType.PROPERTY)
@XmlRootElement
@XmlSeeAlso({XYMeasurement.class,StatisticMesurment.class})
```

Kód 3: Anotace pro JAXB

@XmlAccessorType nám určuje serializovatelnost a *@XmlRootElement* nám bere třídu jako základ pro kořenový zápis do XML, kde se jednotlivé prvky budou brát jako childnody pro třídu.

Díky *@XmlSeeAlso* se program podívá do tříd, které dědí ze třídy *Measurement a* při zápisu a čtení bude brát v potaz jejich instanční proměnné. Takto JAXB zná základ pro další zpracování. Dále se pro některé proměnné používá anotace *@XmlTransient,* která schová danou proměnnou či metodu před JAXB a vůbec ho nebere v úvahu v dalším zpracování. Toto je někdy potřeba, když nechceme některé údaje zapisovat nebo když se JAXB snaží zanořovat stále hlouběji v programu, kam nechceme. Pro některé proměnné jsem použil anotaci *@XmlElement*, která upřesní pro JAXB, jak má zapsat daná data do XML.

Samotné ukládání je implementováno následovně :

```
JAXBContext context =   JAXBContext.newInstance(Measurement.class);
Marshaller marshaller = context.createMarshaller();
Measurement us  = this;
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
marshaller.marshal(us, w);
```

Kód 4: JAXB marshaller

JAXB řekneme, jakou třídu budeme převádět a pak pomocí marshalleru provedeme samotný převod. Díky metodě *setProperty()* můžeme ještě nastavit formát při ukládání.

Dalším krokem bylo upravení metody *openMeasurement()* ve třídě *MainFrame*. V této metodě je implementace celého načítaní jak pro *XYMeasurement,* tak *StatisticMesurment*. Zde se XML převádí zpět na JAVA objekty. U této možnosti se musí marshalleru napsat, jaké třídy bude očekávat (viz kód 5). V případě rozšíření programu se jen připíše další třída, kterou má JAXB zpracovat.

Ve výjimce JAXB je ponecháno staré zpracování přes SAX parser, kdyby někdo chtěl načítat ještě měření ve starém formátu.

```
JAXBContext.newInstance(XYMeasurement.class,StatisticMesurment.class);
```

Kód 5 : otetření měření pomocí JAXB

3.2) Úpravy třídy MainFrame

Třída Mainframe je veškerý základ pro práci s daty a zobrazení programu. Je zde řešeno načítaní a ukládání měření, funkce menu, nastavení měření a jiných parametrů programu. Pro naše potřeby jsem si musel vytvořit nový *JMenuItem*, kterým mohu vytvořit nové *Statistické měření*. Tento item jsem přidal do menu vedle vytváření XY měření. Item má přiřazenou vlastní Akci. Akce je objekt jakékoliv třídy, jenž implementuje rozhraní Action. Toto rozhraní deklaruje metody, které pracují s nějakým akčním objektem. Výhoda tohoto použití je ve spolupráci s komponenty typu *JMenu* nebo *JToolBar,* které mají metodu *add()* a můžou si jednotlivé reakce na akce lehce přidávat.

Vlastní Action dědí ze třídy AbstractAction. V Action můžeme nastavit pomocí metody *putValue()* některé vlastnosti (viz kód 6). Nejdůležitější částí Action je metoda *actionPerformed(),* ve které určíme, co má Action po reakci udělat, například po stisknutí tlačítka.

```
public ActionNewFileDB(){
            super("Nové Statistické měření");
            setEnabled(true);
            putValue(Action.SHORT_DESCRIPTION, "Vytvoří nové měření.");
            putValue(Action.MNEMONIC_KEY, KeyEvent.VK_N);
            putValue(Action.ACCELERATOR_KEY,
      KeyStroke.getKeyStroke("control N"));
            putValue(Action.ACTION_COMMAND_KEY, "NewMeasurement");

           putValue(Action.DISPLAYED_MNEMONIC_INDEX_KEY, 0);
```

```
            putValue(Action.LARGE_ICON_KEY,
                    new
ImageIcon(getClass().getResource("/resource/icons/newfile_wiz.gif")));
            putValue(Action.LONG_DESCRIPTION, "Vytvoří nové měření.");
            putValue(Action.SMALL_ICON, new
ImageIcon(getClass().getResource("/resource/icons/newfile_wiz_small.gif")))
;
        }
        public void actionPerformed(ActionEvent e) {
            int type = MeasurementFactory.MEASUREMENT_STATISTIC;
             createNewMeasurement(type);
        }
```

Kód 6 : Ukázka Action

Akce pro vytvoření Statistického měření předá metodě *createNewMeasurement()* spravný typ pro vytvoření. Tento typ získáme díky třídě MeasurementFactory, ve které je enumerátor se všemi typy měření, se kterými může program UNISAVE pracovat. Action reagující na vytvoření XY měření byl také upraven, aby metodě pro vytvoření měření předával jeho správný typ.

3.2.1) Volba typu ukládaní programu

Základní požadavek pro práci byl, aby program UNISAVE měl možnost volby, zda se bude načítaní a ukládání realizovat pomocí XML nebo přes Hibernate. Proto jsem ve třídě *MainFrame* vytvořil *JcheckBoxMenuItem,* díky kterému mohu měnit nastavení proměnné *SavetoDB* ve třídě *GlobalSettings* a tím říct programu, jak se má chovat co se týče načítaní a ukládaní. Tento CheckBox jsem dal do menu s nastavením. Při jeho inicializaci se rovnou nastaví jeho stav podle údaje, který zjistí z proměnné SaveToDB a poté mu nastaví *Actionlistener*, který bude reagovat na stisknutí checkboxu. Tím se změní proměná *SaveToDB* a typ ukládaní programu. (viz kód 7).

```
jMenuSettings.addSeparator();
ComboMenuItem = new JCheckBoxMenuItem("Ukladani do Databáze");
ComboMenuItem.setMnemonic(KeyEvent.VK_C);
jMenuSettings.add(ComboMenuItem);
ComboMenuItem.setSelected(GlobalSetting.getSavetoDB());
ComboMenuItem.addActionListener(this);
```

Kód 7 : CheckBoX pro proměnnou savetoDB

Díky proměnné *SaveToDB* můžeme pak v celé třídě upravit metody pro ukládaní a načítaní programu, kde do podmínky vložíme typy ukládaní a podle hodnoty v *SaveToDB* se nám provede jedna z nich. Tento způsob jsem použil v metodach, které pracují s databází a XML. Jedná se o metodu, která otevírá měření *openMeasurement()* a metodu, která ukládá měření *savemeasurement()*.

Pro potřeby programu jsem upravil metodu *SaveAsMeasurement()*, která má teď možnost ukládat do XML i v případě, kdy jsme si v programu nastavili, že máme ukládat pouze do databáze.

Dále jsem do třídy *Mainframe* přidal menuAction pro podporu vytváření Reportu, který budu používat v další části práce (viz kod 8). Samotné menuActions se načítají při spuštění programu a mají různé funkce jako kopírovaní, mazání a jiné. Tyto akce mají přiřazené své prvky, které reagují na stisknutí tlačítek. Action se inicializují v metodě *initializeMenuActions()*. Jednotlivé Actions dědí z rozhraní *MeasurementPanelInterface.*

```
getJButtonReportDesigner().setAction(panelInterface.getActionCreateReportDe
signer());
getJMenuItemProtocolCreate().setAction(panelInterface.getActionCreateReport
Designer());
```

Kód 8 : Panely pro GUI

Ve třídě MainFrame se provedla ještě menší úprava, kdy se po vypnutí programu zavolá metoda pro uložení GlobalSettings, kde jsou uložené důležité údaje pro program, včetně proměnné SaveToDB. Dále jsem přidal krátký kód, který nastaví defaultní cesty k souborům pro vytváření reportů, kvůli možnosti tyto cesty měnit během chodu programu pomocí nastavení v menu.

3.3) Upráva třídy StatisticMesurment

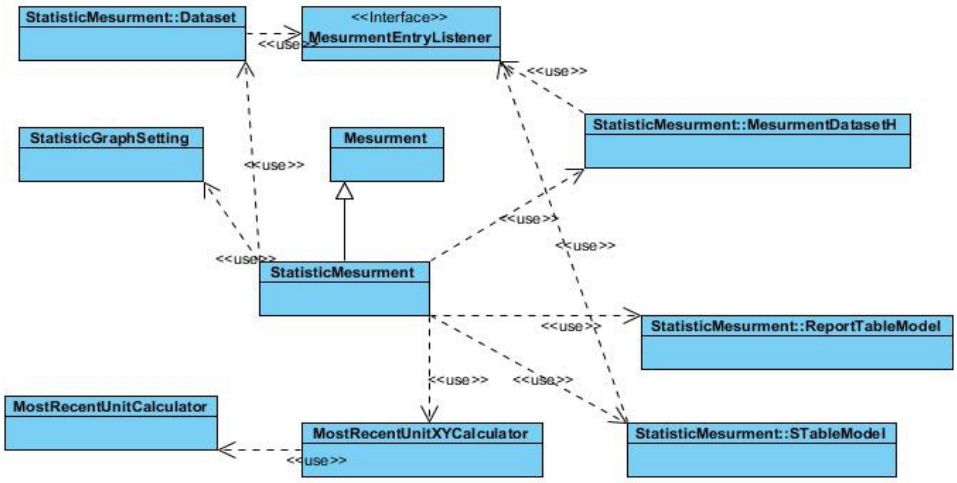
Verze programu, ze které jsem vycházel neměla vůbec implementovanou část pro Statistické měření. Musel jsem ho tedy doplnit z jiného projektu. Během tohoto procesu se objevilo hodně chyb, které většinou souvisely s odlišným používáním názvů proměnných, tříd nebo importů. Zároveň s touto třídou se musely doplnit související třídy, které se staraly o zobrazení uživatelského rozhraní nebo práci s grafy a jejich posluchači.

Základní třídou datové části pro Statistické měření je *StatisticMesurment*. Tato třída se stará o veškerou logiku a uchovávání dat. Tato třída má čtyři vnořené třídy, které nebylo potřeba upravovat. Jedná se o třídy *StatisticMesurmentReportTableModel, STableModel, StatisticMesurmentDataset a StatisticMesurmentDatasetH.*

Třída *StatisticMesurmentDataset* slouží jako datový zdroj k vytvoření grafů. Obsahuje všechny důležité metody pro tvoření grafů. Třída *StatisticMesurmentDatasetH* je podobná předchozí třídě, jen se stará o vykreslení histogramu. Třída *STableModel* obsahuje nastavení samotné tabulky, ve které se zobrazují naměřená nebo vložená data. Třída *StatisticMesurmentReportTableModel* se používá při samotném vytváření protokolu. Obsahuje dvě pole, která slouží pro správné určení dat, jež se mají vepsat do protokolu při jeho vytváření. Jedno pole určuje datové typy a druhé, jaká data se zobrazí při vytvoření protokolu. Díky použití nové knihovny pro vytváření protokolu nebude tato třída dále používáná.

Při použití třídy *StatsticMesurment* jsem musel přepsat veškerá volání metody *setMesurement()*. Program totiž používá metodu *setMeasurement()*. Důležité bylo upravení importů tříd, protože v původní verzi se nacházely v jiných balíčcích a umístěních. Upravil jsem i volání vzdálených metod třídy *GlobalSettings*, kde třída *StatisticMesurment* pracovala trochu jinak. Upraveny byly i některé názvy proměnných a metod.

Třída *StatisticMesurment* používala pro své jednotky datový typ *UnitInterface*, který v mé verzi neexistuje, proto jsem všechny výskyty tohoto použití přepsal nebo přetypoval na použití Unit.



Obrázek 4 : Třídní diagram StatisticMesurment

3.3.1) Přepsání metod pro práci se statistikými hodnotami

Při zkoušení programu se objevil problém u zobrazení statistických hodnot. Tato chyba souvisela s voláním proměnné *MostRecentUnit*, která by měla vracet nejpoužívanější jednotku. Kvůli nekompatibilitě nebylo však tohle možné, proto jsem přepsal následující metody. Jedná se o metody na zjištění nejmenší hodnoty, největší hodnoty, aritmetického průměru a směrodatné odchylky. Upravil jsem tyto metody tak, aby nemusely brát v potaz *MostRecentUnit a* tím nevznikal problém při zobrazení. Upravený kód jedné z metod můžete vidět zde (viz kód 9)

```
double odchylka = 0;
int count = data.size();
if(count < 2)
{
      count = 2;
}
double pred = (double)1/(count);
double sum = 0;
double prum = getAritPrumer();
for(Statistic e : data)
{
      sum +=Math.pow(e.getValueSta().getTrueValue(),2);
}
```

```
odchylka = Math.sqrt((pred * sum)- Math.pow(prum, 2));
int temp = (int)((odchylka+0.00005)*10000.0);
odchylka = ((double)temp)/10000.0;
return (double)odchylka;
```

Kód 9 : Výpočty statistických hodnot

3.4) Úpravy třídy StatisticMesurmentPanel

Tato třída se stará o zobrazení uživatelského rozhraní a předání hodnot měření a jejich úpravu. Tato třída volá při vytvoření několik *JPanelů*. Třemi základními jsou *JPanelMesurement*, který zobrazí základní okno, kde se zobrazují naměřené hodnoty, statistické hodnoty a výběr grabberů pro vkládání hodnot. Dalším je *JPanelGraph*, který zobrazí nastavení pro vytváření grafu a posledním je *JPanelProtocol*, který se stará o zobrazení nastavení protokolu. Tento panel jsem musel upravit pro použití zároveň s databází. Proto jsem změnil volání některých panelů. Tyto panely jsou upravené tak, že mají možnost vkládat data z databáze. Jedná se o uložené laboratoře, měřitele a zákazníky. (viz kód 10). Původní panely měly zamknutou možnost vkládat údaje ručně, toto jsem změnil kvůli možnosti vkládat data i bez použití databáze v rámci výběru uživatele. Podobné úpravy jsem provedl i ve třídě *XYMeasurementPanel*, která se stará o zobrazení pro XY hodnoty.

```
jPanelProtocol.add(getCustomerPanel(), gridBagConstraints4);
jPanelProtocol.add(jLabelObjectOfMesurment, gridBagConstraints5);
jPanelProtocol.add(getJTextFieldObjectOfMesurment(),
                   gridBagConstraints6);
jPanelProtocol.add(jLabeldateOfMesurment, gridBagConstraints7);
jPanelProtocol.add(getDateSelector(), gridBagConstraints8);
jPanelProtocol.add(getEmployeePanel(), gridBagConstraints10);
```

Kód 10: Úprava protokol panelu

3.5) Úpravy souvisejících tříd

3.5.1) MeasurementPanelFactory

Tato třída se stará o vytváření jednotlivých instancí podle předaného typu. Chová se podle návrhového vzoru Factory. V této metodě nebyly dodělány implementace pro Statistické měření, takže jsem ho doplnil.

```
case MeasurementFactory.MEASUREMENT_STATISTIC:
p = new StatisticMesurementPanel();
break;
```

Kód 11: Připsání možnosti pro FactoryMethod

3.5.2) WaitOnGetReport

Tato třída zobrazuje informativní zprávu při vytváření protokolu a potom následného zobrazení protokolu. Zde též chyběla implementace pro Statistické měření. Jedná se o set a get metody pro předání typu měření a úprava metody *run()*, kde je podmínka, která určuje jaký typ protokolu se zobrazí (viz kód 12)

```
if (measurement != null) {
    report = measurement.getReport();          }
if (statMesurment != null) {
      report = statMesurment.getReport();      }
setVisible(false);
```

Kód 12 : Pomínka pro výběr měření

3.6) Úpravy třídy MeasurementDAO

Tato třída je hlavní třídou, která se stará o zpracování dat a jejich uložení či nahrání do databáze. Třída dědí z *AbstractHibernateDAO* a má několik základních metod pro komunikaci s databází. V původní verzi tato třída uměla pracovat jen s XY měřením, toto jsem musel změnit, abych mohl ukládat i statistické měření.

3.6.1) Ukládání měření

První metodou je *SaveOrUpdate()*. Této metodě se předává parametr funkce typu Measurement. V tomto parametru jsou uložena veškerá data, která budeme ukládat do databáze. Původně tento parametr předával vždy jen XY měření, teď ale může obsahovat i statistické měření. Naštěstí pro zjištění, který typ měření byl předán slouží proměná *type,* jejíž hodnotu získám přes metodu *getType()*. Ta mi vrátí číslo typu měření a tak zjistím, které měření bylo předáno. Díky lehké podmínce mohu následně rozdělit zpracování podle typu. Před samotným uložením do databáze se hodnoty ještě předávají do proměných třídy *StatMeasurementPOJO*, která slouží jako prostředník. Předávání hodnot a následné uložení vypadá následovně: (kód 13)

```
StatisticMesurment m = (StatisticMesurment)measurement;
GlobalSetting.getUserSetting().getLastUsedProtocolSetting().
updateFrom(m.getProtocolSetting());
GlobalSetting.getUserSetting().getLastUsedGraphSetting().
updateFrom(m.getGraphSetting());
      StatMeasurementPOJO saved = new StatMeasurementPOJO();
      saved.setAutoConvert(m.getAutoConvert());
      saved.setAutoConvertUnit(m.getAutoConvertUnit());
      saved.setAutoIndex(m.getAutoIndex());
      saved.setGraphSetting(m.getGraphSetting());
      saved.setProtocolSetting(m.getProtocolSetting());
      saved.setXValueName(m.getXValueName());

      try {
      startOperation();
      getSession().save(saved);
      getTransaction().commit();
      } catch (HibernateException e) {
```

```
} finally {
closeSession();
}
m.setModified(false);
}
```

Kód 13 : Ukládání do databáze

Abych mohl ukládat do databáze, bylo potřeba vytvořit mapovací soubor pro *StatMeasurementPOJO*, který určuje jaké proměnné se ukládají a jakým způsobem. Nejdůležitější částí v tomto souboru je změna diskriminátoru, který rozliší o jaký typ měření se jedná. Poté napíšu správné property a typ jejich uložení. Viz kód.

```
<subclass name="StatMeasurementPOJO" extends="MeasurementPOJO"
discriminator-value="STAT">
<property name="XValueName"></property>
<property name="XMarkerFixed"></property>
<many-to-one name="YAutoconvertUnit" class="unisave2006.units.Unit"
cascade="save-update"></many-to-one>
<many-to-one name="graphSetting" class="GraphSetting" cascade="save-
update"></many-to-one>
```

Kód 14: Mapovací soubor StatisticMesurement

Záznam o existenci tohoto mapovacího souboru jsem musel ještě zapsat do základního souboru pro nastavení Hibernate, aby věděl, podle čeho má zpracovat hodnoty.

Při ukládání Statistického měření vznikl problém při ukládání naměřených hodnot. Tyto hodnoty byly typu Statistic a neměly implementovanou podporu pro ukládání do databáze. Bylo nutné přidat metody a proměnné do třídy Statistic, které určovaly indexy pro naměřené hodnoty, aby databáze věděla pořadí prvků. Také bylo potřeba poupravit mapovací soubor pro třídu Statistic. Proměnná *indexXY* se stará o indexaci při ukládání a načítaní.

```
<property name="statType"></property>
<property name="indexXY"></property>
<many-to-one name="valueSta" class="Value" cascade="save-
update"></many-to-one>
<many-to-one name="measurement"
class="unisave2006.data.MeasurementPOJO" column="MSM_ID"></many-to-one>
```

Kód 15: Mapovací soubor třídy Statistic

3.6.2) Načítání měření

Načítaní měření se provádí ve dvou metodách. Jedna je *findAll()*, která najde všechny záznamy v databázi a ty zobrazí v panelu. Tato metoda byla tvořená jen pro XY měření, proto jsem musel provést menší úpravu. Hibernate naštěstí dokáže rozlišovat potomky tříd a proto bylo řešení jednoduché, stačilo napsat v querry, že hledáme nadřazenou třídu *MeasurementPOJO*, ze které dědí *StatMeasurementPOJO* a *XYMeasurementPOJO*.

```
            startOperation();
query = getSession().createQuery("from"+ MeasurementPOJO.class.getName());
            entries = (List<MeasurementPOJO>) query.list();
            getTransaction().commit();
```

Kód 16 : Načtení z databáze pomocí findAll()

Další metodou je *find()*, která vrací právě jedno měření podle id, které bylo předané jako parametr při výběru v panelu. Zde se muselo zjišťovat, jestli třída volaná z databáze je typu *StatMeasurementPOJO* nebo *XYMeasurementPOJO*. Díky pár podmínkám se zjistí typ a vrácená třída se přetypuje dle potřeby. Poté se ještě určí, která proměnná není prázdná a ta se bude z POJO class zpětně plnit daty do odpovídajích tříd *StatisticMesurement* nebo *XYMeasurement*.

```
Object result = getSession().load(MeasurementPOJO.class, id);
if (result instanceof StatMeasurementPOJO)
 {

pojo1 = (StatMeasurementPOJO) result;
      } else if (result instanceof XYMeasurementPOJO) {
            pojo2 = (XYMeasurementPOJO) result;
      } else {
            throw new RuntimeException("Unknown type", null);
      }
```

Kód 17 : Načtení z databáze pomocí find()

3.7) Upravení třídy GlobalSettings

V této třídě se nachází metody pro ukládání a načítaní dat. Jde o názvy meřících zařízení, komunikační data grabberů, informace o jednotkách, poslední nastavení programu a základní nastavení při spuštění. V metodách, které se starají o načítaní nastavení se udělaly úpravy, aby se rozhodovalo zda se mají data načítat z databáze nebo XML.

Původně *GlobalSettings* neměl možnost ukládat svoje hodnoty do XML. Proto jsem vytvořil metodu *SaveGlobalSettings()*, která uloží třídu pomocí JAXB. Zde se ukládají cesty k šablonám, maximální počet posledně otevřených měření, název commBridgeExe, který slouží pro komunikaci se zařízeními a mnou vytvořená proměnná *savetoDB*, kterou program teď používá pro rozlišení zda má pracovat s databází nebo XML.

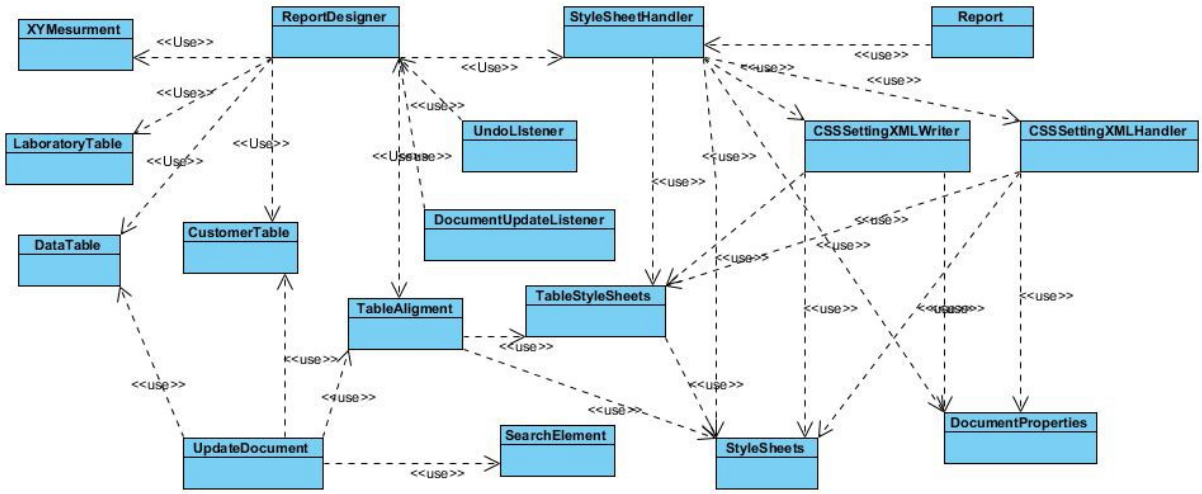
Při načítání používám technologii DOM, protože vznikl problém při předávání hodnot, kdy u JAXB nešlo dobře pracovat s voláním statických metod. Třída by se musela celá přepisovat. Proto jsem zvolil toto řešení.

Zbývající data načítaná z XML stále používají způsob zpracování SAX. Snažil jsem se tento způsob převést na JAXB, ale postupně vznikalo hodně problémů, které se projevovaly v programu špatným a nepřesným načítáním hodnot. Některé chyby byly způsobené odlišnými způsoby načítání a porovnání. Nebylo možné používat jeden typ metod pro oba způsoby ukládání. Program ale i přes to funguje správně buď přes XML nebo JAXB.

3.8) Upravení Report Designeru

Výchozí program UNISAVE neměl naimplementovaný Report Designer, který slouží pro vytváření protokolů. Musel jsem ho tedy do projektu přidat a upravit importy a některé názvy metod a proměných abych mohl Report Designer použít.

Hlavní třídou je *ReportDesigner*. Slouží k zobrazení GUI. Jako textová komponenta je použita *JTextPane*. Tato komponenta se používá jako model pro třídu *HTMLDocument*. Jako "editor kit" je použita třída *HTMLEditorKit*. Pomocí akcí je nadefinováno jak budou jednotlivá tlačítka v menu reagovat na události.



Obrázek 6 : Třídní diagram Report Designeru

3.8.1) Úprava vkládání dat

Pro naše učely jsem musel změnit, co se bude zapisovat do dokumentu po stlačení tlačítek. Původně se zapisovala data z měření mezi HTML tagy. To jsem ale nahradil jen textovým výpisem, co se bude zobrazovat a do tagů jsem napsal identifikaci pro následné zpracování při převedení do XML. Tyto identifikátory se shodují s názvy v tabulce pro zpracování protokolu, aby program následně mohl jednoduše tyto názvy přečíst a vytvořit protokol. Také jsem přidal akce reagující na jednotlivá data v měření. Původně se některá data zapisovala v tabulce. Díky této úpravě má uživatel možnost vše zapisovat jednotlivě.

Po vytvoření dokumentu umí Report Designer vše uložit do HTML souboru. Takto si můžeme přes prohlížeč prohlédnout co jsme vytvořili. V programu je ještě možnost tento HTML soubor změnit na XML soubor, který se dá použít pro vytvoření protokolu. Kvůli použití nové knihovny pro zpracování protokolu není možné použít starý mapovací soubor, který program UNISAVE používal.

3.8.2) Převod Html souboru na XML

Třída *Report* je základní třída, která se stará o převod HTML souborů Report Designeru na XML. Funguje tak, že otevře HTML soubor a nahrazuje některé HTML tagy a přepíše je speciálními XML tagy, které dokáže knihovna pro vytváření protokolu zpracovat.

V metodě pro převod se otevře dokument HTML a začne se po řádku procházet. Pokud narazí na hledané tagy, tak je přepíše a vepíše nové údaje. Do této části bylo nutné přidat reakce na nové tagy, které používají jednotlivá data přidaná v Report Designeru. Vyřešil jsem tento problém přidáním

podmínky reagující na začátek tagu span, pomocí metod pro práci se stringem jsem si vybral jen část uvnitř tagu a tu použil pro naplnění identifikátoru, který se následně zapsal. Pro tuto část jsem si vytvořil pár proměných, které obsahují string se začáteční a koncovou částí. Identifikátor se zapíše mezi ně. Tímto způsobem se projde celý dokument a veškeré HTML tagy se převedou na speciální XML tagy.

```
if(line.contains("<span class=\""))
                {
int begin = line.indexOf("\"");
int end = line.indexOf("\"",(begin+1));
MeasurementId = line.substring((begin+1),end);
if(MeasurementId != null)
                        {
bufferedWriter.write(MeasurementBegin);
bufferedWriter.write(MeasurementId);
bufferedWriter.write(MeasurementEnd);
                        }
MeasurementId = null;
                }
```

Kód 18 : Přepsání stringu

3.8.3) Převod XML na HTML

Dalším požadavkem byl zpětný převod XML souboru pro protokoly zpět na HTML kvůli možnosti náhledu, jak výsledný protokol bude vypadat. Bylo tedy nutné ve třídě ReportDesigner vytvořit Action, která otevírá soubor XML, ten převádí na námi požadovaný HTML soubor, který až poté předává programu a zobrazuje. Nejdříve bylo nutné vytvořit novou Action, která by spustila metodu pro otevření dokumentu. (kód 19)

```
iIcon = new imageIcon(getClass().getResource("/resource/icons/open.gif"));
            actionOpenReport = new AbstractAction("Otevřít Report", iIcon)
            {
            public void actionPerformed(ActionEvent e) {
                  if (!promptToSave())
                  return;
```

```
        openDocumentReport();
        }
};
item = new JMenuItem(actionOpenReport);
item.setMnemonic('o');
item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
InputEvent.CTRL_MASK));
jMenuFile.add(item);
```

Kód 19 : Otevření souboru xml

Tato Action po reakci spustí metodu *openDocumentReport()*, která otevře XML dokument, převede ho na HTML dokument a zobrazí jej. Nejdůležitější částí v této metodě je předání objektu File, který předávám jako parametr metodě *returnReportFile()* ve třídě *Report*, která vrací upravený inputstream, jenž můžu pak otevřít.

3.8.4) Převod XML do HTML pomocí metody returnReportFile()

Tato metoda se stará o celý převod XML souboru na HTML. Stavebním kamenem je dvourozměrné pole, které uchovává v jednom sloupci identifikátory a ve druhém sloupci na stejném indexu vrací data z měření. Metoda prochází všechny řádky a nahrazuje je odpovídajícími HTML znaky. Nejdůležitější je část, kdy se narazí na řádek s identifikátorem. Díky tomu, že víme jak řádek vypadá, si můžeme z dokumentu vybrat pouze identifikátor a ten použít pro porovnání s polem hodnot. Když identifikátor souhlasí s některým z řádků prvního sloupce pole, tak vrátí hodnotu druhého sloupce na tom samém řádku. Tím se nám do HTML dokumentu zapíše hodnota z měření, které bylo předáno hlavní třídě při vytváření ReportDesigneru.

```
For (int i = 0; i < pole[0].length; i++)
{
if (name.trim().equalsIgnoreCase(pole[0][i]))
      {
      return pole[1][i];
      }
}
```

Kód 20 : Metoda pro naplnění dokumentu daty

3.8.5) Upravení třídy SettingDialog a vytváření protokolů

Pro účely programu jsem ještě vytvořil v této třídě několik tlačítek, díky kterým můžu načíst cestu k vlastnímu reportu, který použiju pro vytváření protokolů. Při načtení přepíšu cesty k protokolům novou cestou. Při ukončení programu se cesty k souborům nastaví na původní.

Bylo ještě nutné upravit způsob vytváření náhledu protokolů v celém programu, kvůli nekompatibilitě knihoven. V původních metodách jsem změnil způsob vytváření objektu JFreeReport a jeho plnění daty, knihovny JFreeReport verze 0.9.2+ jsou nekompatibilní se staršími verzemi.

```
JFreeReportBoot.getInstance().start();
String in = GlobalSetting.getReportXYFileName();
ResourceManager manager = new ResourceManager();
manager.registerDefaults();
Resource res = null;
res = manager.createDirectly(in, JFreeReport.class);
final JFreeReport report = (JFreeReport) res.getResource();
report.getInputParameters().put("title",
protocolSetting.getMeasurementTitle());
```

Kód 21 : Vytváření protokolu

3.8.6) Vzhled mapovacího souboru a náhled protokolů

Použitím JFreeReport verze 0.9+ můžeme oddělit strukturální formátování textu od grafického formátování. Definováním CSS stylů lze oddelit strukturu od vzhledu. Mezi elelementy <report::inline-stylesheet> se vyskytují CSS pravidla, která definují vzhled stránky. Dále si můžeme všimnout značek <H1>, které známe z html. Zpracování protokolů podporuje různé html tagy, pro úpravu formátu.

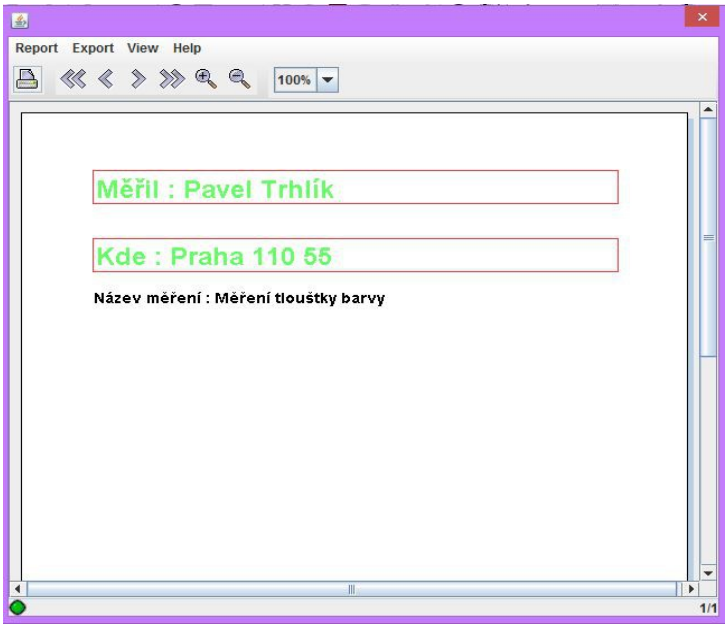
```
<report:report xmlns=”http :// www.w3.org/1999/xhtml” xmlns:report=”http://jfreereport .sourceforge.
    net/namespaces/reports/flow”>
    <report:query>default</report:query>
    <report: inline−stylesheet>
    @page {
    margin−left: 2cm;
    margin−right: 2cm;
    margin−top: 2cm;
    margin−bottom: 2cm;
```

```
size : A4;
}
{ font−family: ” Arial ” ;}
h1 {
border−width: 2px;
border−color: red;
border−style: solid ;
text−align: center;
}
</report: inline−stylesheet>
<h1>Toto je nadpis protokolu</h1>
Obyčejný text ...
Nadpis měření.
<report:content>
<report:value-expression formula="jfreereport:[title]"/>
</report:content>;
</report:report>
```

Kód 22 : Ukázka souboru report.xml

Protože jazyk XML je tzv. Metajazyk, lze u něj definovat vlastní značky, kterým můžeme přiřadit vlastni požadované CSS styly. Je ovšem nutné aby každá značka byla párová.

Za elementem, který uzavírá definici CSS pravidel, již následuje samotný obsah dokumentu, který bude zpracovávat parser knihovny JFreeReport. Kliknutím na možnost vytvořit protokol, nám program zpracuje zadaný report.xml a vytvoří protokol podle zadaných dat.



Obrázek 6 : Protokol JFreeReport

4 Závěr

Jedním z cílů této bakalářské práce byla změna způsobu načítání a ukládání dat pro program Unisave. Původní způsob byl řešen jen pomocí databáze. Ukládání do databáze bylo řešeno přes Hibernate, se kterým jsem se předtím nesetkal a musel jsem tedy nastudovat aspoň základy práce s tímto prostředím. Možnost ukládat do XML byla zakázána. Postupně jsem vracel programu možnost ukládat i do XML. Kde to bylo možné, použil jsem technologii JAXB. Dále bylo nutné přidat programu možnost pracovat se Statistickým měřením. Musel jsem přidat všechny potřebné třídy a upravit je, aby dokázaly pracovat se zbytkem programu. Šlo většinou o upravení názvů metod a proměnných, importů tříd a změny GUI. Ve Statistickém měření bylo nutné změnit celou část pro zobrazení údajů protokolu, aby dokázal zobrazovat data z databáze. Poté jsem přidal ještě podporu pro ukládání do databáze, kde bylo nutno přepsat několik metod pracujících s Hibernate a připsání mapovacích souborů.

Druhou částí bakalářské práce bylo upravení editoru pro vytváření šablon. Editor neměl plnou podporu pro vytvoření šablon ve správném formátu a se správnými daty. Musel jsem tedy vytvořit nová tlačítka společně s objekty Actions, které zapisovaly nové správné údaje do souboru. Dále jsem upravil způsob zápisu při ukládání a vytváření XML report souboru. Přidána byla ještě možnost zpětně otevřít XML soubor a zobrazit údaje v programu, kvůli možnosti náhledu vytvořeného protokolu.

Díky bakalářské práci jsem si rozšířil své znalosti jak v Javě samotné, tak s prací v oblasti Action objektů, tvorby GUI, prací s Hibernate, vytvářením XML souborů pomocí JAXB, vytvářením protokolů pomocí JfreeReport. Tato práce může v budoucnu sloužit jako návod pro implementaci dalších možností ukládání a modulů, či zlepšení práce s vytvářením protokolů.

Reference

[1] HEROUT, Pavel. *Učebnice jazyka Java*. 5., rozš. vyd. České Budějovice: Kopp, 2010, 386 s. ISBN 978-80-7232-398-2.

[2] Syntaxe HTML. [online]. [cit. 2013-07-22]. Dostupné z: http://www.jakpsatweb.cz/html/

[3] How to Use Actions. [online]. [cit. 2013-07-22]. Dostupné z: http://docs.oracle.com/javase/tutorial/uiswing/misc/action.html

[4] Jak na Javu. [online]. [cit. 2013-07-22]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=244

[5] How to Use menus. [online]. [cit. 2013-07-22]. Dostupné z: http://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

[6] Hibernate Examples. [online]. [cit. 2013-07-22]. Dostupné z: http://www.tutorialspoint.com/hibernate/hibernate_examples.htm

[7] XML. [online]. [cit. 2013-07-22]. Dostupné z: http://www.w3schools.com/xml/

[8] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Vyd. 1. Překlad Bogdan Kiszka. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.

[9] KISZKA, Bogdan a Ila NEUSTADT. *1001 tipů a triků pro jazyk Java: objektově orientovaná analýza a návrh prakticky*. Vyd. 1. Překlad Bogdan Kiszka. Brno: Computer Press, 2009, 542 s. ISBN 978-80-251-2467-3.

[10] JAXB tutorial. [online]. [cit. 2013-07-22]. Dostupné z: https://jaxb.java.net/tutorial/

Seznam příloh

Obsah CD

K této práci je přiložené CD – ROM medium, které obsahuje následující soubory:

• Text této bakalářské práce ve formátu PDF

• Zdrojové kódy programu Unisave (export z Eclipse)